LSI LASTEM S.r.l.
Via Ex S.P. 161 Dosso, n.9 - 20090 Settala Premenugo (MI) - Italia

Tel.: (+39) 02 95 41 41
Fax: (+39) 02 95 77 05 94
e-mail: info@lsi-lastem.it

WEB: http://www.lsi-lastem.it
CF./P. Iva: (VAT) IT-04407090150
REA: 1009921 **Reg.Imprese:** 04407090150

RCCL

CISS Communication Library
User Manual

Updating 15/9/2008

Index

1. Introduction....3
2. Product installation....3
3. Example Application....4
4. Fast Start....5
4.1. Preparation of test project....5
4.2. Communication channel....5
4.3. Communication manager....6
4.4. Data reception....7
4.5. During communication....8
4.6. Disposing of the objects....9

1.Introduction

RCCL library is a software that allows the communication with LSI LASTEM instruments that are compatible with CISS proprietary protocol, particularly with E-Log V2 instruments and following releases.

The library can be used to process Windows applications that need to transfer the data produced by E-Log acquirer and CISS compatible instruments. Using library you can obtain this performance easily and fast.

Library has been developed in Microsoft .NET Framework V3.0 and for this reason it can be used by compatible development systems.

2.Product installation

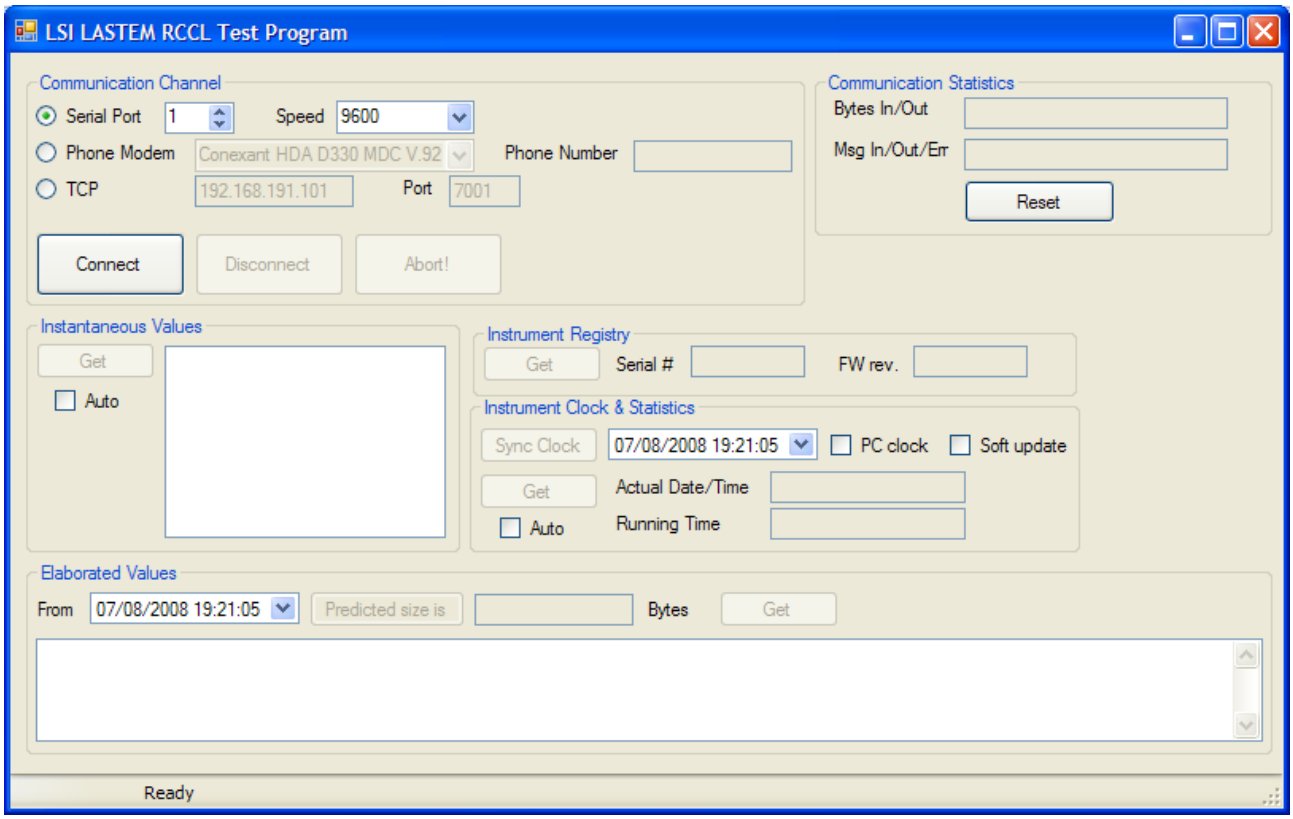
Executing proper installation program Setup.exe you can install the product. Default installation is made in folder "%ProgramFiles%\LSI-Lastem\CISS Communication Library". In this folder are set up following subfolders:

- *bin*: it includes the library's files; particularly:
  - *LSI.RCCL.dll* it's main library's, it's compatible with.NET 3.0;
  - *LSIModemHlp.dll* it's a standard Microsoft COM library (*Common Object Model*) for the management of the modem through TAPI interface; this library must be set correctly using the utility RegSvr32 or other equal system, in order to can be used in user's application; setup procedure supplied by LSI LASTEM made it automatically;
  - *Interop.LSIMODEMHLPLib.dll* it's the library for inter-operation between COM and .NET using library *LSIModemHlp.dll*.
- *doc*: it includes this document;
- *src*: it includes the sources of example application;
- *exe*: example application already filled and ready to-use.

3.Example Application

The library is supplied together with an example application, that includes source code in C# language and has been developed through Microsoft Visual Studio 2008 in .NET 3.0 environment.

The application has following mask:



In order to communicate with an E-Log instrument, use the application following these instructions (it's better if inside development environment, in order to understand its operation):

1) Select the communication channel (Communication Channel) used for connection to the instrument (there are three channels available); set up the use parameters according to used channel;
2) Execute connection of communication channel pressing key *Connect* ;
3) Ask the instrument for instantaneous values pressing key *Get* in the section *Instantaneous Values*; check in the section *Communication Statistics* the steady data stream; at the end of communication the list of instantaneous value should show all measures programmed in the instrument and corresponding last acquired or calculated value; in this section, it's possible communicate with the instrument automatically and continuously sticking box *Auto*;
4) Ask for registry information of the instrument pressing key *Get* in the section *Instrument Registry*;
5) Ask for statistic information about instrument's operation pressing key *Get* of the section *Instrument Clock & Statistics*; in this section, it's possible communicate with the instrument automatically and continuously sticking box *Auto*; furthermore pressing key *Sync Clock* it's possible synchronize the clock inside the instrument with the same date/hour displayed near the key; the synchronization is made in *soft* way (gradually, but only for differences not more than one hour) if it's been stuck box *Soft update* before enter the command; at last

stick box PC clock in order to obtain the automatic updating of text for hour selection according to date/hour of PC's system;

6) Press key *Predicted size is* in the section *Elaborated Values* in order to ask for evaluation of dimension of processed data which need to be transferred, according to specified starting date/hour; so press key *Get* in the same section in order to ask for elaborated data; at the end of communication the pertinent text box should include the received data.

4.Fast Start

Making reference to example application here below you can find the information for implementation of one program which is able to communicate with E-Log instrument and download the acquired data, in fast way; you can find further details about library in proper documentation included in file RCCL.chm. The library has been developed with *C#* language and the examples for its use will have same language.

The examples included in the documentation aren't a complete programming course, at the opposite they need a quite well programming knowledge and ability in .NET environment with language C#.

4.1.Preparation of test project

Set up new application *Windows Form* in development environment; get onto it the reference to LSI.RCCL.dll library.

In the main window of application now you can add the instruction for reference to classes of objects which will be used in succession:

```
using LSI_Lastem.Lib2.Communications2;
using LSI_Lastem.Lib2.CISS2.BaseServices;
using LSI_Lastem.Lib2.CISS2.ELogComm.Data.V2;
using LSI_Lastem.Lib2.CISS2.ELogComm.V2;
```

4.2.Communication channel

Library is able to use serial lines, modem or TCP connections; first of all specify it which communication channel it has to use. This example is using serial connection. To set up serial communication channel we can use a configuration structure type `CommunicationParameters`, where are specified both use parameters of communication channel (channel type, bit rate, serial port number, etc.), and the parameters of communication protocol towards the instrument (network address, number of connection attempts, timeout answer waiting, etc.). If user want use different types of communication channels he needs to set up different values for items of this structure; anyway the management of the communication will be the same (for all type of used channel).

```
CommunicationParameters cp = new CommunicationParameters();
cp.SetElogDefaults();
```

The statement `SetElogDefaults` sets up (all in one go) all correct default parameters for use of the protocol towards E-Log instrument. For this reason you must set up the remaining parameters, in this case the use of one serial nr. 1 with bit rate 9600 bps:

```
cp.ccp.CommChannelType = CommChannelTypes.Serial;
cp.ccp.ComPort = 1;
cp.ccp.Speed = 9600;
```

Now it's possible use following instruction for set up of communication channel:

```
ccb = CommChannelBase.Factory(cp);
```

The object `ccb` is defined on class scale, because it must remain instanced for all its lenght:

```
private CommChannelBase ccb;
```

It'll be advisable to know the connection or disconnection status of communication channel; for this purpose it'll be generated an event that can be linked with the statement:

```
ccb.ConnectedEvent += new
CommChannelBase.ConnectedEventHandler(ccb_ConnectedEvent);
```

Now it's possible ask the communication channel to run connection (serial port opening):

```
ccb.Connect();
```

4.3.Communication manager

The connection will happen receiving the event according to method `ccb_ConnectedEvent`. This method will set up (if it hasn't been already done in other part of the program) the object of class `ELogCommManager`, which performs the main function of the program, that is the communication with the instrument; the object will be set up through structure `cp.pcp` (previously set up), or setting up a local structure, as needed:

```
ProtocolCommunicationParameters pcp = new ProtocolCommunicationParameters();
pcp.SetElogDefaults();
ecm = new ELogCommManager((CommChannelBase)sender, pcp);
```

The object `ecm` includes several capacities that are acquired partly by hierarchy of classes from which it derivates and partly by methods it includes. There are some available facilities that need to know (for their correct operation) the current configuration of parameters that manage the operation of the instrument (setup of: measures, processings, etc.); these information are required by functions for decoding of data acquired and elaborated by measures. `ELogCommManager` is able to ask the instrument the configuration data independently and only if needed, but this operation is very expensive in terms of time and data flow (think about costs in case of use of telephone communication channels). For this reason it's possible specify to `ELogCommManager` to use an

management object of configuration's persistence which is able to utilize all information already received in a previous communication session and already stored in local system (PC), and it doesn't need to download them again from the instrument.

A standard implementation of persistence management object is already available in library RCCL; it's available like class `BasicConfigInfosPersister`, and an object like this type is sent to ELogCommManager through the statement:

```
ecm.ConfigInfosPersister = new BasicConfigInfosPersister();
```

The management of the object `BasicConfigInfosPersister` by means of `ELogCommManager` is totally clear, because the configuration data are persistent in PC and they are read again only if needed and automatically.

At the end you have to connect to events of object `ELogCommManager`, in order to manage the facilities in asynchronous way:

```
ecm.DataReceivedEvent += new
CommHelper.DataReceivedEventHandler(ecm_DataReceivedEvent);
ecm.BinDataReceivedEvent += new
CommHelper.BinDataReceivedEventHandler(ecm_BinDataReceivedEvent);
ecm.CommSequenceTerminatedEvent += new
ELogCommManager.CommSequenceTerminatedEventHandler(ecm_CommSequenceTerminatedEve
nt);
```

4.4.Data reception

Now everything is ready to start communication; for example, in order to receive the data from the instrument (last values acquired or calculated by measures), it's possible use method:

```
ecm.GetCalcValues();
```

Method `GetCalcValues`, and all methods for request of data RCCL implemented, is asynchronous, (i.e. its call finishes fast, before the end of communication with the instrument). In this way it's possible manage the interface of application without problems (stops), but it's necessary to limit the operations allowed to user before the end of communication (for example: avoid the call to other communication method, because the library could have an exception type *Protocol is busy*).

At the end of communication there are two possibilities: the communication has ended successfully or the communication has failed. In first case it'll be possible use the received data, in the second case the user needs to know the reason of failure, so that he could solve the problem. In this case the problem has to be checked through method linked to event `CommSequenceTerminatedEventHandler`.

In order to use the received data, input following code in method linked to event `DataReceivedEvent`:

```
CISSCalcValues ccv = (CISSCalcValues)e.DataReceived;
```

The received data are sent to method through an object coming from `EventArgs`, which includes an object coming from `CISSBaseData` and that's dedicated to include the data transferred into specific communication with the instrument. In case of instantaneous data of measures it's type `CISSCalcValues`; this object has simple methods and properties for recovery of measures' values.

Same system for use of other data transferred in ASCII mode by CISS protocol; among these data there are the operation statistic data of the instrument (`CISSStatistics`), the registry data (`CISSRegistry`), and other data. The processings of the measures, stored in instrument's memory, are transferred through binary code (because they are potentially very big), for this reason it's necessary take an other system .

The events stored in `ecm` object include also type `BinDataReceivedEvent`. The method linked to this event is able to use the data received through binary way, like processings. At beginning it's necessary ask the library to run the communication with the instrument in order to receive the processings. The instrument is able to store big quantity of data and for this reason it isn't convenient receive all contents every-time. So it's possible to specify to RCCL library that it has to receive the data starting from specified date/hour and arriving to last datum available in the instrument, the latest datum. For this reason use method:

```
ecm.GetElaborations(fromDT);
```

The object `fromDT` is type `DateTime` and includes the date/hour of first required processing. The method that instrument uses for data storage could start some processings shortly before the specified date, but it always assures the sending of data processed from required date/hour and the next data. The manage of the data through the application has to take into consideration this possibility and for this reason filter the data received but not required.

At the end of communication the method linked to event `BinDataReceivedEventHandler` will be recalled, and it will be possible to use the received data:

```
List<ElabDataRow> edrc = (List<ElabDataRow>)e.DataReceived;
```

The processings are included in one list of object `ElabDataRow`, every object represent one processing event and includes one or more elaborated items. For example, if the instrument has been configured with two measures and every 10 minutes each measure produces statistic processings type *Min*, *Ave*, *Max*, each object `ElabDataRow` will include 6 numeric values corresponding to one processing event of 10 minutes, of which is also specified date/hour, sorted like configured in the instrument, i.e. *Misura1(Min*, *Ave*, *Max), Misura2 (Min*, *Ave*, *Max)*.

4.5.During communication

Some communications need lot of time to be concluded, especially if they use slow communication channels. In order to assure the user about the right operation of the application it's possible to supply the statistic of running communication in real time. It can be reached through the object that includes the communication channel; the property `ecm.CommChannelLayer.Statistics` includes useful information for this purpose, and they can be easily show through application's user interface.

4.6.Disposing of the objects

The objects that include the communication's channel and protocol can keep in use valued resources of the system, i.e. the serial communication lines. After their use we recommend to dispose them through available method *Dispose*. Before the disposing of the object we recommend to disconnect previous connected events, in order to avoid the reception, because they could have been started by asynchronous facilities of the library.

```
ecm.DataReceivedEvent -= new
CommHelper.DataReceivedEventHandler(ecm_DataReceivedEvent);
ecm.BinDataReceivedEvent -= new
CommHelper.BinDataReceivedEventHandler(ecm_BinDataReceivedEvent);
ecm.CommSequenceTerminatedEvent -= new
ELogCommManager.CommSequenceTerminatedEventHandler(ecm_CommSequenceTerminatedEve
nt);
ecm.Dispose();

ccb.ConnectedEvent -= new
CommChannelBase.ConnectedEventHandler(ccb_ConnectedEvent);
ccb.Dispose();
```